MacApp 2.0 Object and Method Reference

This chapter describes the object classes that existed in MacApp 1.1. If you need information on classes and methods not described here, refer to other release notes and to the source code.

Each object description in this chapter contains the following elements:

- whether you customize the object type, instantiate it, or call its methods
- notes about the object type
- the chain of ancestors leading to the object type
- field declarations and explanations
- descriptions of the methods for each object type

Important

Complete information about the implementation of each method is not given in this chapter. If you need further details about any method, refer to the MacApp source code.

TObject

Customize: usually
Instantiate: never
Call methods: usually

TObject is the ultimate ancestor for all objects in MacApp.

TObject is documented here primarily for background information. It is an abstract object type that exists so that other object types can inherit characteristics from it, and thus share them.

The only TObject methods you might override are Free and Clone.

Ancestors: none

Fields

none

Clone

```
FUNCTION TObject.Clone: TObject;
```

The return value	An exact copy of the calling object
Purpose	To clone dependent objects referred to by the fields of an object as well as cloning the object itself. An object is dependent on another object when the second object has the only (or the only important) reference to the first object. Dependency is a relatively vague condition; when you override this method, you need to determine what objects are dependent on SELF.
The default version	Calls ShallowClone, and thus clones only the object itself
Override	Sometimes
Call	Sometimes

Free

```
PROCEDURE TObject.Free;
```

Purpose	To free the calling object and any dependent objects referred to by its fields. An object is dependent on another object when the second object has the only (or the only important) reference to the first object. Dependency is a relatively vague condition; when you override this method, you need to determine what objects are dependent on SELF.
The default version	Calls ShallowFree
Override	Often. Your version should free any dependent objects you have added for your customization and then call INHERITED Free so that any ancestor methods can free other dependent objects. The chain of INHERITED calls leads to TObject.Free, which calls TObject.ShallowFree, which frees SELF.
Call	Often

ShallowClone

```
FUNCTION TObject.ShallowClone: TObject;
```

The return value	text
Purpose	This is the lowest-level method for copying an object.
Called by	TObject.Clone
The default version	Calls HandToHand, an Inside Macintosh routine, to copy the object data
Override	Never
Call	Rarely

ShallowFree

```
PROCEDURE  TObject.ShallowFree;
```

Purpose	This is the lowest-level method for freeing an object.
Called by	TObject.Free
The default version	Frees the calling object by calling DisposHandle
Override	Never
Call	Rarely

TEvtHandler

Customize: rarely
Instantiate: never
Call methods: sometimes

TEvtHandler is documented here primarily for background information. It is an abstract object type that exists so that other object types can inherit characteristics from it, and thus share them.

The primary importance of TEvtHandler is that it allows the different objects that handle events to be stored in a single list.

Ancestors: TObject

Fields

`fIdlePriority: INTEGER;` A priority value for the DoIdle method of this object. If fIdlePriority is not greater than zero, the default Idle method never calls this object's DoIdle method. The default Idle method calls the DoIdle methods of any handlers with fIdlePriority values greater than zero. (The default value is 0.)

`fNextHandler: TEvtHandler;` The next handler in the chain of event handlers, or NIL

`fIdleFreq: LONGINT;` Defines the minimum number of ticks (each tick = 1/60th of a second) that must elapse before this object's DoIdle gets called. A value of zero means that DoIdle gets called as often as possible (assuming that the object instance is in the target chain or cohandler chain). A value of kMaxIdleTime means the object's DoIdle never gets called. The default value is kMaxIdleTime

`fLastIdle: LONGINT;` The tick at which this object's DoIdle method was last called.

DoHandleEvent

```
FUNCTION TEvtHandler.DoHandleEvent(nextEvent: PEventRecord; VAR commandToPerform:
TCommand): BOOLEAN;
```

nextEvent	A pointer to the new event
commandToPerform	A command object that will perform the action indicated by the event or gNoChanges, if the action has already been done or the event resulted in no command
The return value	Indicates whether or not the event has been handled

Purpose	To handle an alien event
Called by	TApplication.HandleAlienEvent. TApplication.HandleAlienEvent implements the cohandler chain.
The default version	Returns FALSE
Override	You always override this method when you create a cohandler. A cohandler is an event handler that is not in the target chain and is not a view, window, document, application, print handler, or command object. You create cohandlers to handle alien events, which are generally asynchronous events like network events. Your implementation of DoHandleEvent should return TRUE if it handles the event and, otherwise, return FALSE.
Call	Never

DoIdle

```
PROCEDURE TEvtHandler.DoIdle(phase: IdlePhase);
```

phase	Whether idle is just beginning, is continuing, or is ending. The declaration of IdlePhase is `IdlePhase = (idleBegin, idleContinue, idleEnd);`

Purpose	To do idle-time tasks. This method is called for event handlers only when `fIdleFreq` ticks have elapsed—but only when the handler is in the target or cohandler chain.
Called by	TApplication.Idle
The default version	Does nothing
Override	When an object requires idle-time processing.
Call	Never

DoKeyCommand

```
FUNCTION TEvtHandler.DoKeyCommand(ch: CHAR; VAR info: EventInfo): TCommand;
```

ch	A character typed at the keyboard
info	The event information record that contains the key event. You can modify this parameter if you want.
The return value	A command object

Purpose	To handle "key commands," which are simply events resulting from keyboard typing
Called by	TApplication.ObeyEvent
The default version	Calls DoKeyCommand for the next handler in the list of event handlers. If there is no next handler, the default method returns gNoChanges.
Override	Sometimes. If you override this method, generally for your descendant of TView or TDocument, you should return a command object that can respond appropriately to the character. (See "TCommand" in this chapter for more information.) For simple editing, this method is implemented in the TEView unit. (See the "Using TEView" recipe in the Cookbook chapter or the TTETypingCommand section of this chapter for more information.)
Call	Sometimes. You call this method if you override it, by calling INHERITED DoKeyCommand. Otherwise, you never call it.

DoMenuCommand

```
FUNCTION TEvtHandler.DoMenuCommand(aCmdNumber: CmdNumber): TCommand;
```

aCmdNumber	The command number for the menu command chosen by the user
The return value	A command object or, if there are no changes, gNoChanges

Called by	TApplication.MenuEvent when a menu command is chosen by the user. (A Command-key combination is usually equivalent to a menu command.)
The default version	Calls fNextHandler.DoMenuCommand if there is a next handler. If there is no next handler, the default returns gNoChanges and, if the code was compiled with debugging on, prints an error message.
Override	Often. You override this method to handle menu commands you have defined. In general, you return a command object to carry out the action of the command; if the command is simple and does not change the document, you can return gNoChanges.
Call	Often. You call this method if you override it, by calling INHERITED DoMenuCommand. Otherwise, you never call it.

DoMultiClick

```
FUNCTION TEvtHandler.DoMultiClick(lastDownPt, newDownPt: Point): BOOLEAN;
```

lastDownPt	The next-to-last point where the mouse button was pressed
newDownPt	The most recent point where the mouse button was pressed
The return value	TRUE if lastDownPt and newDownPt are close enough to be considered a double click

Purpose	To test whether a new mouse click should be counted as an additional click in gClickCount. It should return TRUE if newDownPt is close enough to lastDownPt to be considered the same point.
Called by	TApplication.CountClicks
The default version	Calls fNextHandler.DoMultiClick if there is a next handler. If there is no next handler, it tests whether the difference between the two points is less than or equal to five pixels.
Override	Rarely. If you want to change the standard for what is considered a new multiple click, you can override TApplication.DoMultiClick. The default version always calls fNextHandler.DoMultiClick unless fNextHandler is NIL, which is true only for the application object.
Call	Never

DoSetupMenus

```
PROCEDURE TEvtHandler.DoSetupMenus;
```

Purpose	To adorn and enable (or disable) all menu commands handled by this event handler. This method is called before menus are displayed when the menus may have changed since the last time it was called. It is also called after every event is processed.
Called by	TApplication.SetupTheMenus and when an immediate descendant's method calls INHERITED DoSetupMenus
The default version	Calls DoSetupMenus for the next event handler in the list of event handlers. (TEvtHandler is not responsible for any menu commands.)
Override	Sometimes. You must override this method if you define any menu commands. In general, you override this method for any object types for which you override DoMenuCommand, and you handle the same menu commands in DoMenuCommand and DoSetupMenus for a given object type. When you override this method, you must begin your method by calling INHERITED DoSetupMenus, so that MacApp can set up the menus first. Then, you use the global procedures Enable and EnableCheck to enable any menu commands that can currently be used or to disable any that cannot be used. (EnableCheck, like Enable, can enable or disable menu commands. EnableCheck also can add or remove a check mark next to a menu item.) You can also adorn menus in other ways. See the "Changing Menu Appearance and Function" recipe in the Cookbook for more detailed information.
Call	Sometimes. You always call this method when you override it.

IEvtHandler

```
PROCEDURE TEvtHandler.IEvtHandler(itsNextHandler: TEvtHandler);
```

itsNextHandler	The next handler in the list of event handlers, or NIL
Called by	TApplication.IApplication, TDocument.IDocument, TView.IView, and TPrintHandler.IPrintHandler to initialize an event-handler object
The default version	Sets the value of fIdleFreq to kMaxIdleTime, sets fLastIdle to zero, and sets fNextHandler to itsNextHandler
Override	Never
Call	You call this method only if you declare immediate descendants of TEvtHandler.

Terminate

```
PROCEDURE  TEvtHandler.Terminate;
```

Purpose	To handle termination tasks for an event handler
The default version	Does nothing
Override	Sometimes
Call	Never. The TApplication and TPrintHandler implementations of this method are called by MacApp.

TApplication

Customize: always
Instantiate: never
Call methods: always

The application object controls the overall application. In other words, it implements methods that apply to the application as a whole rather than to an individual document or window.

You always customize TApplication to implement your application.

Ancestors: TObject, TEvtHandler, TApplication

Fields

`fIdleFreq: LONGINT;`	Defines the minimum number of ticks (each tick = 1/60th of a second) that must elapse before this object's DoIdle gets called. A value of zero means that DoIdle gets called as often as possible (assuming that the object instance is in the target chain or cohandler chain). A value of kMaxIdleTime means the object's DoIdle never gets called. The default value is kMaxIdleTime
`fNextHandler: TEvtHandler;`	The next handler in the chain of event handlers, or NIL. Inherited from TEvtHandler.

* **Note:** Other fields are inherited but are never used. TApplication declares no new fields. Many global variables are used like fields of the application object.

AboutToLoseControl

```
PROCEDURE TApplication.AboutToLoseControl;
```

Called by	TApplication.HandleSystemEvent, TApplication.PostHandleEvent (when the user clicks in a nonapplication window), and TApplication.Run (just before the end)
The default version	Commits the last command and writes the contents of the Clipboard to the desk scrap (if necessary)
Override	Sometimes. You can override this method to do other tasks necessary before the application loses control.
Call	Never

AddFreeWindow

```
PROCEDURE TApplication.AddFreeWindow(aWindow: TWindow);
```

aWindow	A window object
Purpose	To add a window to the free window list. A free window is one that belongs to the application instead of to a document. (An example is the palette window in MacPaint.)
Called by	TWindow.InstallDocument
The default version	Calls gFreeWindowList.AddLast(aWindow)
Override	Never
Call	Rarely

ClaimClipboard

```
PROCEDURE TApplication.ClaimClipboard(clipView: TView);
```

clipView	The Clipboard view created to show the Clipboard contents
Called by	The application to insert the given view in the Clipboard
Override	Rarely
Call	You always call this method for a Cut or Copy command, unless you don't implement cutting and pasting in your application. See "The Clipboard" in the Cookbook for more information.

CloseWmgrWindow

```
PROCEDURE TApplication.CloseWmgrWindow(aWmgrWindow: WindowPtr);
```

aWmgrWindow	The Window Manager pointer for a window that is being closed
Called by	TApplication.DoMenuCommand (if the user choose the Close command) and TApplication.HandleMouseDown (if the mouse press was in the close box) and TApplication.Close (when the application is terminated).
The default version	Checks whether the window is a desk accessory window and, if it is, calls CloseDeskAcc (an Inside Macintosh procedure) to close it. If it is not a desk accessory window, this method checks whether there is a window object for this window and, if there is, calls its CloseByUser method. Otherwise, it calls HideWindow (an Inside Macintosh procedure). It signals failure with err = 0 if the user cancels for some reason.
Override	Rarely
Call	Never

CommitLastCommand

```
PROCEDURE TApplication.CommitLastCommand;
```

Called by	TApplication.AboutToLoseControl, TApplication.CheckDeskScrap, TApplication.PerformCommand, TDocument.Close, TDocument.Revert, and TDocument.Save.
The default version	Commits and frees the last command (gLastCommand) and changes the text for the Undo command to show these there is no current undoable command
Override	Rarely
Call	Never

CountClicks

```
FUNCTION TApplication.CountClicks(aPDownEvent: PEventRecord): INTEGER;
```

aPDownEvent	A pointer to the event record for a mouse-down event
The return value	The current number of multiple clicks

Called by	TApplication.HandleMouseDown
The default version	Calls gTarget.DoMultiClick to see whether the new mouse press should be considered an additional multiclick. If so, it increments gClickCount. Otherwise, it resets gClickCount to 1.
Override	Rarely
Call	Never

DeleteFreeWindow

```
PROCEDURE TApplication.DeleteFreeWindow(windowToDelete: TWindow);
```

windowToDelete	A member of the free window list

Purpose	To remove a window from the free window list. A free window is one that belongs to the application instead of to a document. (An example is the palette window in MacPaint.)
Called by	TWindow.Free and TWindow.InstallDocument
The default version	Calls gFreeWindowList.Delete(windowToDelete)
Override	Never
Call	Rarely

DoCommandKey

```
FUNCTION TApplication.DoCommandKey(ch: CHAR; VAR info: EventInfo): TCommand; OVERRIDE;
```

ch	The character of the key that was held down along with the Command key
info	The event record

Purpose	To handle events in which a key is pressed along with the Command key as a Command-key equivalent of a menu command
Called by	TApplication.HandleKeyDownEvent
The default version	Calls SetupTheMenus and MenuEvent unless this is a repeated command-key combination or if gRepeatcmd is FALSE
Override	Rarely. You can override this method to implement your own Command-key commands or to implement key commands in your own way. Note that you need do nothing with this method for Command-key combinations that correspond to menu commands and are given in the resource file. MacApp does not implement auto-key events (automatic repeating of keys held down) with Command-key combinations. If you want to implement auto-key Command-key combinations, you must override this method.
Call	Never

DoMakeDocument

```
FUNCTION TApplication.DoMakeDocument(itsCmdNumber: CmdNumber): TDocument;
```

itsCmdNumber	Indicates the type of document that should be created. In applications with different document types, the command number indicates which menu command the user picked or, if the user opened an existing document, the command number is the one returned by TApplication.KindOfDocument.
The return value	A document object

Purpose	To create a document for the application. It is called when the user starts up the application, opens a document with the New or Open command, and in other cases when the application needs to create a document.
Called by	TApplication.OpenNew, TApplication.OpenOld, and TApplication.PrintDocument
The default version	Calls ProgramBreak to halt the program. (You must override this method.)
Override	Always. Your implementation of this method creates and initializes a document of your application's type. If your application has multiple document types, your implementation of this method creates different document types depending on the value if itsCmdNumber. See the "Creating a Document" recipe in the Cookbook for details on this method.
Call	Sometimes. You may call this method to create a document, but most commonly, this method is called by MacApp.

DoMenuCommand

```
FUNCTION TApplication.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
```

aCmdNumber	The command number of the menu command chosen by the user
The return value	A command object that will carry out the command (and possibly undo and redo the command) or gNoChanges

Purpose	To handle menu commands that apply to the application as a whole
Called by	TApplication.MenuEvent when there is a menu command and gTarget is a reference to the application object, or by another object's DoMenuCommand method when no other object has handled the menu command
The default version	Handles the MacApp defined standard menu commands Quit, New, Open, Close, Undo, Redo, ShowClipboard, About <Appname>, and the Debug menu commands
Originally declared by	TEvtHandler
Override	Often. You override this method to handle commands you define for your application object. In general, TYourApplication.DoMenuCommand handles the commands that apply to the application as a whole. When your implementation does not handle the command, you should end your override method by calling INHERITED DoMenuCommand, so that the MacApp method can handle its commands.
Call	You always call this method if you override it. Otherwise, you never call it.

DoSetupMenus

```
PROCEDURE TApplication.DoSetupMenus; OVERRIDE;
```

Purpose	To set up the menu commands handled by the corresponding DoMenuCommand method. It is called before the menus are displayed when they may have changed since the last time DoSetupMenus was called.
The default version	Sets up menu commands handled by TApplication.DoMenuCommand
Originally declared by	TEvtHandler
Override	Often. You must override this method if you override TApplication.DoMenuCommand. Your override method must set up the menu commands handled by TYourApplication.DoMenuCommand. Begin your method by calling INHERITED DoMenuCommand so the MacApp methods can set up their menu commands first. See the "Changing Menu Appearance and Function" recipe in the Cookbook for more details.
Call	You always call this method if you override it. Otherwise, you never call it.

EachFreeWindow

```
PROCEDURE TApplication.EachFreeWindow(PROCEDURE DoToWindow(aWindow: TWindow));
```

DoToWindow	A procedure that will be passed each free window in turn
Purpose	To apply DoToWindow to all windows in the free window list. A free window is one that belongs to the application instead of to a document. (An example is the palette window in MacPaint.)
Called by	Your code and TApplication.ForAllWindowsDo
Override	Never
Call	You might call this method if you have free windows.

ForAllDocumentsDo

```
PROCEDURE TApplication.ForAllDocumentsDo(PROCEDURE DoToDoc(aDocument: TDocument));
```

DoToDoc	A procedure, usually local to the caller, that ForAllDocumentsDo calls repeatedly, passing each of the documents in turn
Purpose	To perform an operation on all documents of an application
The default version	Automatically scans through the list of documents and calls DoToDoc once for each document
Called by	TApplication.AlreadyOpen, TApplication.Close, and TApplication.ForAllWindowsDo
Override	Never
Call	Sometimes

ForAllWindowsDo

```
PROCEDURE TApplication.ForAllWindowsDo(PROCEDURE DoToWind(aWindow: TWindow));
```

DoToWind	A procedure, usually local to the caller, that ForAllWindowsDo calls repeatedly, passing each window in turn

Purpose	To perform an operation on all windows of an application
The default version	Calls DoToWind once for each window of all documents of the application and for any documentless windows
Override	Never
Call	Sometimes

GetDataToPaste

```
FUNCTION TApplication.GetDataToPaste(aDataHandle: Handle; VAR dataType: ResType):
LONGINT;
```

aDataHandle	A handle for Clipboard data
dataType	A data type that is passed back to you
The return value	If nonzero, indicates an error

Purpose	To get data for pasting from the Clipboard
Called by	Your methods
Override	Rarely
Call	You can call this method if you implement the Paste command. You allocate an empty handle, and then pass it to this method. The dataType is not set here; it is one of the resource types you tell MacApp you can handle when you call CanPaste (a MacApp global routine). The data may come from data cut or copied from your application or it may come from the desk scrap. See the "Paste" recipe in the Cookbook for more information.

GetEvent

```
PROCEDURE TApplication.GetEvent(eventMask: INTEGER; sleep: LONGINT; cursorRgn:
RgnHandle; VAR anEvent: EventRecord): BOOLEAN;
```

eventMask	A mask indicating the kind of events wanted
sleep	The minimum number of ticks that cam elapse before returining from WaitNextEvent.
cursorRgn	A region, in global screen coordinates, in which the cursor will not change.
anEvent	The event obtained
The return value	Indicates whether or not an event was obtained

Called by	TApplication.PollEvent and TApplication.UpdateAllWindows
The default version	Calls the Inside Macintosh routine GetNextEvent or WaitNextEvent
Override	Sometimes. You override this method so that you can get events from another source. See Inside Macintosh for information on posting events.
Call	Never

IApplication

```
PROCEDURE TApplication.IApplication(itsMainFileType: OSType);
```

itsMainFileType	The unique four-letter type code for the main document files used by the application

Called by	IYourApplication
The default version	Initializes a number of global variables and otherwise initializes the application
Override	Never. Instead of overriding this method, you generally write a new method with a name of the form IYourApplication.
Call	Always. You always call this method from IYourApplication.

Idle

```
PROCEDURE TApplication.Idle(phase: IdlePhase);
```

phase	The current part of the idle sequence: idleBegin, idleContinue, or idleEnd

Purpose	This method is called when all events have been handled and there are no pending events.
Called by	TApplication.PollEvent and TApplication.GetEvent
The default version	Begins by setting up the menus, if they need to be set up, and then gives each event handler that needs idling a chance to run its DoIdle method
Override	Rarely
Call	Never

InstallCohandler

```
PROCEDURE TApplication.InstallCohandler(aCohandler: TEvtHandler; addIt: BOOLEAN);
```

aCohandler	The handler you want to add to or remove from the cohandler list
addIt	Indicates whether aCohandler should be added to (TRUE) or deleted from (FALSE) the cohandler list

Purpose	To add or remove cohandlers from the cohandler list
The default version	Adds aCohandler to the cohandler list if addIt is TRUE and deletes it if addIt is FALSE. If aCohandler is deleted from the list, it is not freed.
Override	Never
Call	You always call this method if you have cohandlers.

KindOfDocument

```
FUNCTION TApplication.KindOfDocument(itsCmdNumber: CmdNumber; itsPAppFile: PAppFile):
CmdNumber;
```

itsCmdNumber	Either the command number from DoMenuCommand or cFinderNew, cFinderPrint, or cFinderOpen
itsPAppFile	A pointer to an AppFile record or NIL. If not NIL, itsPAppFile^.fileType gives the four-character file type, which is usually all you need to decide what type of document is needed. If NIL, this is a new document, so there is no existing document from which to get information.
The return value	A command number to pass to DoMakeDocument

Purpose	To fix the command number whenever DoMakeDocument is called if the application has more than one kind of document type. It is called by MacApp.
Called by	TApplication.OpenNew, TApplication.OpenOld, and TApplication.PrintDocument
The default version	Returns itsCmdNumber
Override	You always override this method if your application has more than one kind of document. When the user opens an existing document, your implementation of this method uses itsPAppFile to determine what kind of document object should be created. The command number you return is normally the same as the command number for the New menu command the user would choose to create a new document. (In applications with multiple document types, you usually have different New menu commands for different document types.)
Call	Never

LaunchClipboard

```
PROCEDURE TApplication.LaunchClipboard;
```

Called by	TApplication.Run
The default version	Starts up the Clipboard by creating a view and a window for it
Override	Rarely
Call	Never

MainEventLoop

```
PROCEDURE  TApplication.MainEventLoop;
```

Called by	TApplication.Run
The default version	Loops until the application begins to terminate. Events are dispatched from this method and the Idle method is called from this method.
Override	Rarely. You might override this method to change the progress of the event loop. If you do, examine the implementation of TApplication.MainEventLoop in UMacApp.TApplication.p.
Call	Never

MakeViewForAlienClipboard

```
FUNCTION  TApplication.MakeViewForAlienClipboard:  TView;
```

The return value	A Clipboard view
Purpose	To make a view to show the public scrap when the application has just started or has returned from a desk accessory or another application and the desk scrap contains data from another application or from another instance of this application
Called by	TApplication.ReadFromDeskScrap
The default version	Creates a view that can show PICT or TEXT data
Override	Usually. In your implementation, you check the desk scrap to see if it has data in one of the forms your application can handle (presumably because the data came originally from this application or another application that creates compatible data). If data is there in that form, you create a view of one of your application's types to show the data and return that view. Otherwise, you call INHERITED MakeViewForAlienClipboard so that that method can show the PICT or TEXT data. See "The Clipboard" in the Cookbook for more information.
Call	You always call this method if you override it, in which case you call it by using INHERITED.

OpenNew

```
PROCEDURE TApplication.OpenNew(itsCmdNumber: CmdNumber);
```

itsCmdNumber	The command number that resulted in this call
Purpose	To create a new document, including the views and windows for the document. It is called whenever a new document is needed, either when the application starts up or when the user chooses the New command.
Called by	TApplication.DoMenuCommand and TApplication.HandleFinderRequest
The default version	Calls DoMakeDocument, DoInitialState, DoMakeViews, DoMakeWindows, and ShowWindows
Override	Rarely. You may override this method if you don't want a new document to be automatically created when the application starts without the user opening an existing document. To do that, you can test the value of itsCmdNumber. If itsCmdNumber = cFinderNew, you should not create the new document. See the UMacApp source text for other details of the implementation to be certain you do everything necessary.
Call	Never

OpenOld

```
PROCEDURE TApplication.OpenOld(itsOpenCmd: CmdNumber; anAppFile: AppFile);
```

itsOpenCmd	The command number that resulted in this call
anAppFile	An AppFile record
Purpose	This method is called whenever an existing document is opened, either when the application starts up or after the user chooses the Open command.
The default version	Calls DoMakeDocument, DoReadFromFile, DoMakeViews, DoMakeWindows, and ShowWindows
Called by	TApplication.DoMenuCommand and TApplication.HandleFinderRequest
Override	Rarely
Call	Never

PerformCommand

```
PROCEDURE TApplication.PerformCommand(command: TCommand);
```

command	A command object to carry out the most recent command
Purpose	To carry out a command that is not gNoChanges
Called by	TApplication.HandleEvent
The default version	If either the fCanUndo or the fChangesDocument flag is TRUE, TApplication.CommitLastCommand sets gLastCommand to the new command, sets the command's fTarget field to gTarget, sets the command's fCmdDone field to TRUE, and calls command.DoIt. If the command is undoable, the default version puts the command's name in the Undo command. If fChangedDocument <> NIL and fChangesDocument = TRUE, the default version increments the document's change count.
Override	Rarely
Call	Never

PrintDocument

```
FUNCTION TApplication.PrintDocument(anAppFile: AppFile): BOOLEAN;
```

anAppFile	An AppFile record
The return value	Whether or not the document was printed
Called by	TApplication.HandleFinderRequest to handle a Print command from the Finder
The default version	Calls DoMakeDocument, ReadFromFile, and DoMakeViews, telling each that this is being done just for printing, and then calls document.Print for the new document
Override	Rarely
Call	Never

Run

```
PROCEDURE TApplication.Run;
```

Called by	Your main program after you create and initialize your application object
The default version	Does some initialization, calls TApplication.HandleFinderRequest, and then calls MainEventLoop. When MainEventLoop returns, Run calls AboutToLoseControl and CleanUpMacApp.
Override	Rarely. If you want to do something different before calling MainEventLoop, examine the implementation of that method in the UMacApp source to see the details of its implementation. In general, though, it is better to create a different method, and call that before calling Run.
Call	Always

SetUndoText

```
PROCEDURE TApplication.SetUndoText(cmdDone: BOOLEAN; aCmdNumber: CmdNumber);
```

cmdDone	Indicates whether this command is in do or redo phase (TRUE) or in undo phase (FALSE)
aCmdNumber	The command number for the Undo menu command

Purpose	To set the text for the Undo menu command
Called by	TApplication.SetupTheMenus
The default version	Changes the text to Redo if cmdDone = FALSE and back to Undo if cmdDone = TRUE
Override	Rarely
Call	Never

ShowError

```
PROCEDURE TApplication.ShowError(error: OSErr; message: LONGINT);
```

error	An error number
message	A failure message. See the "Failure Handling" recipe in the Cookbook for more information.

Purpose	To display an error message
Called by	Failure handlers
The default version	Calls the global procedure ErrorAlert
Override	Sometimes. You override this method if you want a different error message to be displayed.
Call	Sometimes

SFGetParms

```
PROCEDURE TApplication.SFGetParms(itsCmdNumber: CmdNumber; VAR dlgID: INTEGER; VAR
where: Point;
                        VAR fileFilter, dlgHook, filterProc: ProcPtr; typeList:
HTypeList);
```

itsCmdNumber	The command number that resulted in this method call
dlgID	The resource ID for the dialog box that should be displayed
where	The position of the upper left corner of the dialog box in global coordinates
fileFilter	A pointer to a filter function that determines which files appear in the dialog box, or NIL. If NIL, no filter function is executed.
dlgHook	A pointer to a function that handles dialog items, or NIL. If NIL, no function is executed.
filterProc	A pointer to a function that filters events, or NIL. If NIL, no standard filtering is done.
typeList	A valid handle to a zero-length block

Purpose	To get parameters that should be passed to SFGetFile, which is an Inside Macintosh procedure that displays a dialog box listing files that can be opened by the application
Called by	TApplication.CanOpenDocument and TApplication.ChooseDocument
The default version	Returns these values: dlgID = getDlgID where = (100, 100) fileFilter = NIL dlgHook = NIL filterProc = NIL The typeList parameter returns the main file type supported by the application.
Override	Sometimes. You can override this method to return different parameter values. If the application supports all file types, you should make typeList empty. See Inside Macintosh for more information on the parameters of this method.
Call	Rarely. If you do call this method, you must set typeList to a valid handle and free the handle afterwards.

SFPutParms

```
PROCEDURE TApplication.SFPutParms(itsCmdNumber: CmdNumber; VAR dlgID: INTEGER; VAR
where: Point;
                        VAR prompt, defaultName: Str255; VAR dlgHook, filterProc:
ProcPtr);
```

itsCmdNumber	The command number that resulted in this method call
dlgID	The dialog box that should be displayed
where	The position of the upper left corner of the dialog box
prompt	The prompt string that should be added to the dialog box
defaultName	The default name used in the dialog; it must be initialized to a valid string when this method is called.
dlgHook	A pointer to a function that handles dialog items, or NIL. If NIL, no function is executed.
filterProc	A pointer to a function that filters events, or NIL. If NIL, no standard filtering is done.

Purpose	To return all the parameters that should be passed to SFPutFile
The default version	Returns these values:
dlgID	= putDlgID where = (100, 100) prompt = prompt from resource file dlgHook = NIL filterProc = NIL The defaultName parameter is left alone.
Override	Sometimes. You can override this method to change the default values.
Call	Rarely

TrackCursor

```
FUNCTION TApplication.TrackCursor: BOOLEAN;
```

The return value	Whether a view set the cursor shape.
Purpose	To track the mouse pointer while the mouse button is up
Called by	TApplication.Idle
The default version	Checks the location of the mouse and calls HandleCursor for the window in which the mouse is located
Override	Sometimes. You can override this method to do something else while the mouse button is up. If you do that, you generally call INHERITED TrackCursor.
Call	Never (except by calling INHERITED TrackCursor when you override it)

TDocument

Customize: always
Instantiate: never
Call methods: rarely

The document object controls the data of the document.

Almost every application must define at least one descendant of TDocument for its own document type. The only exception is for "documentless" applications, in which the application icon is always opened.

You generally add fields to your document type to store the views of the document.

If your application has more than one kind of document, you usually create more than one descendant of TDocument, one for each kind of document. For example, an integrated application might have a TTextDocument, a TSpreadSheetDocument, and a TGraphicsDocument type.

Most MacApp applications can have several document objects at a time, which may all be of a single type or may be of different types. The document objects are stored in a TList object stored in gDocList.

Each document object can have one document file. You can use the data and resource forks of the file or use either fork alone. Normally, the entire contents of the file is read into memory when the file is opened, but support is provided for disk-based documents. If the resource fork is used, the document's resource file is on top of the resource file list when DoRead and DoWrite are called. Otherwise, you need to call UseResFile to make sure that the right resource file is on top.

When a document is saved, MacApp normally saves the altered version of the document to a new file and then, when the save operation has been successfully completed, renames the new version of the file, erasing the old version.

A number of the fields of TDocument determine whether the data and resource forks of the file are both opened and how the file is treated when it is saved:

- fDataOpen and fRsrcOpen determine whether or not the data and resource forks of the file should be kept open at all times. Most applications set both to FALSE. An application can have either or both TRUE if the application uses disk-based documents.

Note: Keeping resource files open at all times is usually a bad idea because of the space required for multiple resource maps and the slow searching of multiple files (especially with the 64K ROM). We recommend that keepsRsrcOpen always be FALSE.

- fDataPerm and fRsrcPerm determine what permission is used to open each fork of the file. Each of those can have the values
  - fsRdPerm, for read-only permission
  - fsWrPerm, for write-only permission
  - fsRdWrPerm, for read and write permission
  - fsRdWrShPerm, for shared permission
- fSaveInPlace determines what happens when there isn't enough disk space to save a copy of the file instead of writing over the original. Its values can be
  - sipNever, to indicate that the original file should never be overwritten
  - sipAlways, to indicate that the original file should always be overwritten when there is not enough space for a copy
  - sipAskUser, to indicate that the user should be asked whether or not the original file should be overwritten when there is not enough space for a copy

See the description of IDocument for information on how these fields are initialized.

Programmers who want to implement work files such as MacWrite uses should open them in TYourDocument.IYourDocument and close them in TYourDocument.Free. TYourDocument.FreeData should reset the work file to the same state set by IYourDocument. TYourDocument.DoInitialState should set up the work file for an empty document (if necessary) and TYourDocument.DoRead should set up the work file for an existing document (if necessary). fSaveExists is a reliable indicator of whether a main document file exists or not (and, if fDataOpen or fRsrcOpen is TRUE, whether the corresponding refNum is valid).

Ancestors: TObject, TEvtHandler

Fields

`fChangeCount: LONGINT;` The number of changes since the last time the document was saved

`fCommitOnSave: BOOLEAN;` Whether to commit the last command when saving this document, *if* that command affects the document. The default is TRUE.

`fCreator: OSType;` A four-character code giving the document's creator.

`fDataOpen: BOOLEAN;` Whether or not the data fork of the document file should be kept open at all times. This is FALSE except for disk-based documents

`fDataPerm: INTEGER;` The permission used to open the data fork of the file: fsRdPerm, fsWrPerm, fsRdWrPerm, or fsRdWrShPerm

`fDataRefNum: INTEGER;` The reference number for the data fork of the document file, if that fork is open

`fDocPrintHandler: TPrintHandler;` The object that enables and executes the Print, Print One, and Page Setup commands

`fFileType: OSType;` A four-character code giving the type of the document file

`fModDate: LONGINT;` File modification date representing when the file was last read or saved.

`fPrintInfo: Handle;` Either NIL or a handle to a 120-byte print information record

`fReopenAlert: BOOLEAN;` Whether to give an alert if the user attempts to reopen a document. The default is TRUE.

`fRsrcOpen: BOOLEAN;` Whether or not the resource fork of the document file should be kept open at all times

`fRsrcPerm: INTEGER;` The permission used to open the resource fork of the file: fsRdPerm, fsWrPerm, fsRdWrPerm, or fsRdWrShPerm

`fRsrcRefNum: INTEGER;` The reference number of document file's resource fork, if it is open.

`fSaveExists: BOOLEAN;` Whether or not a disk file representing this document exists; in other words, whether or not this document has ever been saved

`fSaveInPlace: SIPChoice;` The value that determines what happens when there isn't room on the disk to save the document in a new file, rather than writing over the old versionof the document (when the old version is overwritten, the file is "saved inplace"): sipNever, sipAlways, or sipAskUser

`fSavePrintInfo: BOOLEAN;` When this is set to TRUE and the document is saved, TDocument.DoWrite writes the print information record of the fDocPrintHandler to the data fork of the document file. If this is TRUE, when the document is read, the print information record is read by TDocument.DoRead.

`fSharePrintInfo: BOOLEAN;` When this is set to TRUE, all print handlers associated with views belonging to this document will share the same print information record. (This value determines whether or not they will share that record.)

`fTitle: STRING[63];`	The name of the document file
`fUsesRsrcFork: BOOLEAN;`	Whether or not the document uses the resource fork of the file
`fUsesDataFork: BOOLEAN;`	Whether or not the document uses the data fork of the file
`fViewList: TList;`	The list of views that render this document's data
`fVolRefNum: INTEGER;`	The volume reference number of the document file
`fWindowList: TList;`	The list of windows belonging to this document

Close

```
PROCEDURE TDocument.Close;
```

Purpose	To close and free a document. This method must never be called for a document related to a view in the Clipboard.
Called by	TApplication.Close and TWindow.CloseByUser
The default version	If the document's data has changed, a dialog is posed asking the user to save changes. If the user cancels nothing further happens. If the user chooses yes the document's Save method is called and, if necessary, the last command is committed, all of the document's windows are closed, and the document is freed.
Override	Sometimes
Call	Sometimes

DoInitialState

```
PROCEDURE TDocument.DoInitialState;
```

Called by	MacApp methods when the user chooses the New command, when the user chooses the Revert command and there is no saved file, and when the user opens the application icon. It does any additional initialization of the document that is not done when an existing document is opened.
The default version	Does nothing
Override	Often. You should override this method when new documents need initialization not done when existing documents are opened.
Call	Never

DoMakeWindows

```
PROCEDURE  TDocument.DoMakeWindows;
```

Purpose	Primarily, to maintain compatibility with MacApp 1.x by providing the ability to create window objects for a document. This method is called after a document is opened, initialized, and has its views created. This method should create the windows and frames to show the views.
Called by	TApplication.OpenNew and TApplication.OpenOld
The default version	Does nothing.
Override	Sometimes. In your implementation, you may wish to distinguish between views that represent windows and views that represent data.
Call	Never

DoMakeViews

```
PROCEDURE  TDocument.DoMakeViews(forPrinting:  BOOLEAN);
```

forPrinting	Tells you whether or not MacApp called this in response to the user requesting printing of a document from the Finder. If your application creates views that are not printed (such as palette views), you do not need to create them when forPrinting is TRUE.

Purpose	To create the windows and views for a document, both the views that interpret the document's data and those, like palettes, that are independent of the data. It is called after a document is created and initialized and before the windows are created.
Called by	TApplication.OpenNew, TApplication.OpenOld, and TApplication.PrintDocument
The default version	Calls ProgramBreak to halt the program
Override	Always. Your implementation creates all views for the document and stores the views in a field of the document object. See the "Creating a View" recipe in the Cookbook for details on how to implement this method.
Call	Never

DoMenuCommand

```
FUNCTION TDocument.DoMenuCommand(aCmdNumber: CmdNumber): TCommand; OVERRIDE;
```

aCmdNumber	The command number of the menu command chosen by the user
The return value	A command object that will carry out the command (and possibly undo and redo the command) or gNoChanges

Purpose	To handle menu commands defined for this particular object type
Originally declared by	TEvtHandler
The default version	Handles the MacApp-defined standard menu commands Save As, Save a Copy In, Save, and Revert
Override	Often. You override this method when your application has its own menu commands that apply to the document as a whole. In that case, you end your method by calling INHERITED DoMenuCommand so that the MacApp method can handle its commands.
Call	You call this method when you override it. Otherwise, you never call it.

DoNeedDiskSpace

```
PROCEDURE TDocument.DoNeedDiskSpace(VAR dataForkBytes, rsrcForkBytes: LONGINT);
```

dataForkBytes	Indicates the amount of disk space the document needs to save itself. This is set by this method.
rsrcForkBytes	Indicates the amount of disk space the document needs to save itself. This is set by this method.

Purpose	To return the amount of disk space needed to save the document
The default version	Returns 0 for dataForkBytes unless fSavePrintInfo is TRUE, in which case it returns the size of the print information record, and sets rsrcForkBytes to 0 unless fUsesRsrcFork is TRUE, in which case it sets it to the standard fixed overhead value for the resource file. (See the Resource Manager chapter of Inside Macintosh for more information.)
Override	Almost always. Documents that do not override DoNeedDiskSpace generally cannot save any data except the print information record. Your override method should accurately predict how much disk space will be needed to store the data and resources for the documents. (Most documents have no resources, so the resource fork value is usually 0.) When you calculate your values, you do not have to calculate how many blocks are actually needed, just the number of bytes since MacApp automatically accounts for an integral number of blocks. Also, you should add your needs to the initial values of these variables, as MacApp may have already set them to some value before calling this method. If you use the resource fork, you can use the constants kRsrcTypeOverhead and kRsrcOverhead to account for the resource file overhead for each resource type and individual resource, respectively. If there isn't enough space in the target volume, MacApp tests whether deleting the old file would make enough room. If it would, what happens next depends on the value of fSaveInPlace. See the notes at the beginning of "TDocument" in this chapter for more information. If deleting the file would not make enough space (or is precluded by the value of fSaveInPlace or the user's actions), MacApp issues a disk full error and the user is shown an alert to that effect.
Call	Never

DoRead

```
PROCEDURE TDocument.DoRead(aRefNum: INTEGER; rsrcExists, forPrinting: BOOLEAN);
```

aRefNum	A file-system reference number for the document file. It is obtained from the Operating System by MacApp. If the document doesn't use the data fork (that is, it uses only the file's resource fork), aRefNum is 0.
rsrcExists	Indicates whether or not the resource fork of the file exists. If it is FALSE, and the document uses the resource fork, it means that the resource fork could not be opened (presumably because it does not yet exist).
forPrinting	TRUE if the document is being opened just to print it (for printing from the Finder)

Purpose	To read an existing document file so its data can be used in the document object
The default version	Reads the print information record if fSavePrintInfo is TRUE for this document object. Otherwise, it does nothing.
Override	Almost always. Documents that do not override this method cannot save or restore anything except their print information record. If your document uses the resource fork and the resource fork exists, then MacApp will ensure that the topmost resource file is that of the document when this method is called. You may want to get the reference number of the resource file at the start of this method if you think that some other method might change the top resource file. Your implementation generally begins with a call to INHERITED DoRead so that the print information record is read, if necessary. It then reads the data of the document and stores it in fields or objects available to the document object. You should check the rsrcExists parameter before trying to read the resource fork. (It is possible that the user opened a document with no resource fork. MacApp does not consider this an error.) See the "Saving and Restoring Data" recipe in the Cookbook for details about implementing this method.
Call	You call this method if you override it. When you override this method, you usually call it (by calling INHERITED DoRead). Otherwise, you never call it.

DoSetupMenus

```
PROCEDURE TDocument.DoSetupMenus; OVERRIDE;
```

Purpose	To set up menu commands handled by TDocument.DoMenuCommand. This method is called before any menu is displayed when the menus may have changed since the last time it was called or from the idle loop, again when the menus may have changed since the last time this was called. It is responsible for adorning and enabling or disabling all menu commands handled by the document.
Originally declared by	TEvtHandler
The default version	Begins by calling INHERITED DoSetupMenus. It then sets up the menu commands handled by TDocument.DoMenuCommand: Save As, Save a Copy In, Save, and Revert.
Override	Often. You override this method if you define any menu commands that apply to your document. In general, you override this method whenever you override TDocument.DoMenuCommand. Your implementation must begin by calling INHERITED DoSetupMenus so that MacApp can set up the menus first. Then, you use the global procedures Enable and EnableCheck to enable any menu commands that can currently be used. (EnableCheck, like Enable, can enable or disable menu commands. EnableCheck also can add or remove a check mark next to a menu item.) You can also adorn menus in other ways. See the "Changing Menu Appearance and Function" recipe in the Cookbook for more detailed information.
Call	You usually call this method when you override it. Otherwise, you never call it.

DoWrite

```
PROCEDURE TDocument.DoWrite(aRefNum: INTEGER; makingCopy: BOOLEAN);
```

aRefNum	A file-system reference number for the document file. It is obtained from the Operating System by MacApp.
makingCopy	Indicates whether DoWrite is being called to save a copy of the document. (Generally used only for disk-based documents.)

Purpose	To save a document's data to a disk file
The default version	Saves the print information record to the disk file if fSavePrintInfo is TRUE. Otherwise, it does nothing.
Override	Almost always. Documents that do not override this method cannot save or restore anything except their print information record. Your implementation generally begins with a call to INHERITED DoWrite so that the print information record is saved, if necessary. It then saves the document's data. If your document uses the resource fork and the resource fork exists, then MacApp will ensure that the topmost resource file is that of the document when this method is called. You may want to get the reference number of the resource file at the start of this method if you think that some other method might change the top resource file. See the "Saving and Restoring Data" recipe in the Cookbook for details of this method.
Call	You call this method when you override it (by calling INHERITED DoWrite). Otherwise, you never call it.

ForAllViewsDo

```
PROCEDURE TDocument.ForAllViewsDo(PROCEDURE DoToView(aView: TView));
```

DoToView	A procedure, usually local to the caller, that is called repeatedly by ForAllViewsDo and passed each of the views in turn

Purpose	To perform an operation on all views of a document
The default version	Calls DoToView once for each view in the document's view list
Override	Never
Call	Sometimes

ForAllWindowsDo

```
PROCEDURE TDocument.ForAllWindowsDo(PROCEDURE DoToWind(aWindow: TWindow));
```

DoToWind	A procedure, usually local to the caller, that is called repeatedly by ForAllWindowsDo and passed each window of this document in turn

Purpose	To perform an operation on all windows of a document
The default version	Automatically scans through the document's list of windows and calls DoToWind once for each window
Override	Never
Call	Sometimes

FreeData

```
PROCEDURE TDocument.FreeData;
```

Purpose	To free the document's data objects during a revert operation
Called by	TDocument.Revert
The default version	Does nothing
Override	Always. You override this method to free data objects that should be freed when the user chooses the Revert command.
Call	Sometimes. You may want to call this method from your implementation of TDocument.Free, if convenient.

FreeFile

```
PROCEDURE TDocument.FreeFile;
```

Purpose	To free resources associated with the connection between a TDocument object and a disk file
Called by	TDocument.Free, TDocument.SaveViaTemp, and TDocument.SaveInPlace
The default version	Closes the appropriate forks of the file if fDataOpen or fRsrcOpen and fSaveExists are TRUE
Override	Sometimes
Call	Rarely

FreeFromClipboard

```
PROCEDURE  TDocument.FreeFromClipboard;
```

Purpose	To free a Clipboard document
Called by	TView.FreeFromClipboard
The default version	Removes gClipWindow from fWindowList and calls Free
Override	Sometimes. You can override this method to do something other than Free.
Call	Never

GetTempName

```
PROCEDURE  TDocument.GetTempName(VAR  fileName:  Str255);
```

fileName	A name for a temporary document file

Purpose	To generate a random temporary filename
The default version	Appends a mutated form of the time of day to the name of the document or, if the document is untitled, to the name of the application
Override	Rarely
Call	Sometimes

IDocument

```
PROCEDURE TDocument.IDocument(itsFileType, itsCreator: OSType;
      usesDataFork, usesRsrcFork, keepsDataOpen, keepsRsrcOpen: BOOLEAN);
```

itsFileType	The file type for the document file
itsCreator	The signature of the application that created the document file
usesDataFork	Indicates whether (kUsesDataFork) or not (NOT kUsesDataFork) the document uses the data fork of the file
usesRsrcFork	Indicates whether (kUsesRsrcFork) or not (NOT kUsesRsrcFork) the document uses the resource fork of the file
keepsDataOpen	Indicates whether (kDataOpen) or not (NOT kDataOpen) the data fork of the file should be kept open at all times
keepsRsrcOpen	Indicates whether (kRsrcOpen) or not (NOT kRsrcOpen) the resource fork of the file should be kept open at all times

Purpose

To initialize a TDocument object. It is usually called from the initialization method of customizations of TDocument.

The default version

Gives these values to the fields of TDocument:

```
fWindowList := NewList;
fViewList := NewList;
fDocPrintHandler := NIL;
fChangeCount := 0;
fSavePrintInfo := FALSE;
fSharePrintInfo := TRUE;
fPrintInfo := NIL;
fTitle := '';
fFileType := itsFileType;
fVolRefNum := 0;
fReopenAlert := TRUE;
fSaveExists := FALSE;
fCommitOnSave := TRUE;
fCreator := itsCreator;
fDataPerm := fsRdPerm;
fRsrcPerm := fsRdPerm; {Has no meaning with 64K ROM}
fDataOpen := keepsDataOpen;
fRsrcOpen := keepsRsrcOpen;
IF keepsDataOpen OR keepsRsrcOpen THEN
   fSaveInPlace := sipNever
ELSE
   fSaveInPlace := sipAskUser;
```

Override

Never

Call

You call this method at the beginning of the IYourDocument method that you write for your document type to change any values you need to change and do any additional initialization you require.

SavedOn

```
PROCEDURE TDocument.SavedOn(fileName: Str255; volRefNum: INTEGER);
```

fileName	The name of the document file
volRefNum	The volume reference number for the file

Purpose	To allow the programmer to clean up any data structures or work files to note that a clean save has been made
Called by	TDocument.Save when a new copy of the file is being made (the normal situation)
The default version	Resets fChangeCount to 0, sets fSaveExists to TRUE, replaces fTitle and fVolRefNum with the values passed in, and if fDataOpen or fRsrcOpen is TRUE, opens the appropriate fork
Override	Sometimes
Call	Never

SaveInPlace

```
PROCEDURE TDocument.SaveInPlace(itsCmdNumber: CmdNumber; makingCopy: BOOLEAN; VAR
fileName: Str255;
                        volRefNum: INTEGER);
```

itsCmdNumber	The command number for this save operation
makingCopy	Whether or not a copy of the original file is being saved
fileName	The name of the document file
volRefNum	The volume reference number for the file

Purpose	To save the document, replacing the old version on disk
Called by	TDocument.Save when makingCopy is FALSE and askForFileName is FALSE, and the document cannot or should not be saved via a temporary file
The default version	If fDataOpen and fRsrcOpen are both FALSE, deletes the target file, calls SELF.FreeFile, calls SELF.MakeNewCopy, and then calls SELF.SavedOn. If either fDataOpen or fRsrcOpen is TRUE, the default version does nothing.
Override	Sometime. You can override this method to save a disk-based document in place by modifying the file. If you do, you must set the file's access permission to a modifiable mode before doing so.
Call	Never

SaveViaTemp

```
PROCEDURE TDocument.SaveViaTemp(itsCmdNumber: CmdNumber; makingCopy: BOOLEAN; VAR
fileName:
                        Str255; volRefNum: INTEGER);
```

itsCmdNumber	The command number for this save operation
makingCopy	Whether or not a copy of the original file is being saved
fileName	The name of the document file
volRefNum	The volume reference number for the file

Purpose	To save the document into a new, temporary file
Called by	TDocument.Save when a new copy of the file is being made (the normal situation)
The default version	Calls SELF.MakeNewCopy and then calls SELF.FreeFile if makingCopy is FALSE. It then deletes the target (if it exists), renames the file, and calls TDocument.SavedOn if makingCopy is FALSE.
Override	Rarely
Call	Never

SetTitle

```
PROCEDURE TDocument.SetTitle(aTitle: Str255);
```

aTitle	The new title for the window

The default version	Sets SELF.fTitle to aTitle and calls SetTitleForDoc for each window of the document
Override	Sometimes
Call	Sometimes

ShowReverted

```
PROCEDURE  TDocument.ShowReverted;
```

Called by	TDocumentDoMenuCommand when the user chooses the Revert command and clicks the OK button in the dialog box that is displayed
The default version	Calls ShowReverted for each view of the document
Override	Rarely
Call	Rarely

ShowWindows

```
PROCEDURE  TDocument.ShowWindows;
```

Purpose	To display a document's windows on the screen. It is called when the document is initially opened
Called by	TApplication.OpenNew and TApplication.OpenOld
The default version	Calls OpenWindow for all windows for which fOpenInitially = TRUE
Override	Sometimes. You can override this method to determine in some other way what windows are initially shown.
Call	Never

TCommand

Customize: usually
Instantiate: never
Call methods: rarely

TCommand objects fall into two general categories: command objects and mouse trackers. The Cookbook includes a number of recipes dealing with different types of command objects and mouse trackers. In general, you override DoIt, UndoIt, RedoIt, and possibly Commit for command objects and trackers that change the document, while you override TrackConstrain, TrackFeedback, and TrackMouse only for mouse trackers.

Command objects and mouse trackers that do not change the document do not need UndoIt, RedoIt, or Commit. In fact, you may never create a command object for many commands that do not change the document; in those cases, you can carry out the action of the command from DoMenuCommand, DoMouseCommand, DoKeyCommand, DoCommandKey, or another method that returns a command object. (In that case, return gNoChanges.)

Ancestors: TObject

Fields

`fCanUndo: BOOLEAN;` Whether or not this command can be undone. The default is TRUE.

`fCausesChange: BOOLEAN;` Whether or not this command changes the document referred to by the command's fChangedDocument field. This defaults to TRUE. When this is TRUE, the document is automatically marked as changed when this command is done. (If the command is undone, the document's change count is automatically decremented, and if the command is redone, the change count is incremented again.)

`fChangedDocument: TDocument;` The document that may be changed by the command. This defaults to gDocument

`fChangesClipboard: BOOLEAN;` Whether or not this command changes the Clipboard. This defaults to FALSE and should be set to TRUE for cut or copy commands that change the Clipboard.

`fCmdNumber: CmdNumber;` The command number associated with the command

`fConstrainsMouse: BOOLEAN;` When this is set to TRUE, this command's TrackConstrain method is called as the mouse moves. This defaults to FALSE.

`fScroller: TScroller;` Either a handle to the scroller used for auto-scrolling or Nil.

`fTarget: TEvtHandler;` The target to set before calling UndoIt or RedoIt. In other words, the value of gTarget when this command was initially given.

`fTrackNonMovement: BOOLEAN;` Whether to call TrackMouse even if the mouse hasn't moved since the last call to TrackMouse. The default is FALSE.

`fView: TView;` The view in which mouse tracking takes place or Nil to track in screen coordinates.

`fViewConstrain: BOOLEAN;` Whether the mouse is constrained to the view. The default is TRUE.

Commit

```
PROCEDURE  TCommand.Commit;
```

Purpose	To do anything necessary to make the effects of a command permanent
Called by	TApplication.CommitLastCommand, which is called when the command can no longer be undone or redone (usually when a new undoable command is chosen, when the document is closed, or when the application is terminated). It is not called if the command was left undone.
The default version	Does nothing
Override	Often. This method is most commonly used to implement filtered commands or with commands that delete items from the document's data set, in which the deleted items are not freed until the command can no longer be undone.
Call	Rarely

DoIt, RedoIt, UndoIt

```
PROCEDURE  TCommand.DoIt;
PROCEDURE  TCommand.RedoIt;
PROCEDURE  TCommand.UndoIt;
```

Purpose	To do, undo, and redo a command. DoIt is called when the command is initially done; UndoIt is called when the user picks the Undo command an odd number of times; RedoIt is called when the user picks the Undo command an even number of times. DoIt and RedoIt carry out the action of the command (generally, they both call the same methods to do the command, although RedoIt may have to change the selection or otherwise act to restore the state of the document at the time the command was originally done). UndoIt reverses the action of the command.
Called by	TApplication.PerformCommand (DoIt) and TApplication.DoMenuCommand (UndoIt and RedoIt)
The default version	Does nothing
Override	Usually. These are the methods that generally carry out (and undo) the action of the command. The only command objects that may not override these methods are mouse trackers and commands that do not change the document or those that cannot be undone.
Call	Almost never. The only likely exception is that your RedoIt method might call DoIt.

ICommand

```
PROCEDURE TCommand.ICommand(itsCmdNumber: CmdNumber, itsDocument: TDocument; itsView:
TView;
                        itsScroller: TScroller);
```

itsCmdNumber	The command number associated with this command
itsDocument	The document affected by this command
itsView	The view in which mouse tracking takes place or Nil to track in screen coordinates.
itsScroller	Either a handle to the scroller used for auto-scrolling or Nil.

Purpose	To initialize fields of TCommand
Called	Usually from the initialization methods for the immediate descendants of TCommand

The default version	Makes these assignments:
	`fCmdNumber := itsCmdNumber;`
	`fCanUndo := TRUE;`
	`fCausesChange := TRUE;`
	`fChangedDocument := gDocument;`
	`fConstrainsMouse := FALSE;`
	`fViewConstrain := TRUE;`
	`fChangesClipboard := FALSE;`
	`fTrackNonMovement := FALSE;`
	`fView := itsView;`
	`fScroller := itsScroller;`
	`fTarget := NIL;`

Override	Never. You usually supplement its action with an IYourCommand method.
Call	Always. You call this method as part of command initialization.

TrackConstrain

```
PROCEDURE TCommand.TrackConstrain(anchorPoint, previousPoint: VPoint; VAR nextPoint:
VPoint);
```

anchorPoint	The position of the mouse pointer, in view coordinates, when the mouse button went down
previousPoint	The position of the mouse pointer the last time this method was called, in view coordinates
nextPoint	The current position of the mouse pointer, in view coordinates

Purpose	To constrain the mouse movement in any way your application requires. It is used only in mouse trackers.
Called by	TApplication.TrackMouse (a method you never deal with directly) when command.fConstrainsMouse is TRUE
The default version	Does nothing
Override	Sometimes override this method to change the value of nextPoint. See "Handling Mouse Events" in the Cookbook for further discussion of mouse trackers.
Call	Rarely

TrackFeedback

```
PROCEDURE TCommand.TrackFeedback(anchorPoint, nextPoint: VPoint; turnItOn,
mouseDidMove: BOOLEAN);
```

anchorPoint	The position, in view coordinates, of the mouse pointer when the mouse button went down
nextPoint	The current position, in view coordinates, of the mouse pointer
turnItOn	Indicates whether the feedback is to be turned on (TRUE) or turned off (FALSE)
mouseDidMove	TRUE if the mouse moved since the last time TrackFeedback was called

Purpose	To provide on-screen feedback for the user while the mouse is being tracked (that is, while the mouse button is down and a mouse tracker object exists)
Called by	TApplication.TrackMouse
The default version	Provides "rubberband" feedback: a shadowy box between anchorPoint and nextPoint
Override	Often. You override this method to provide more appropriate feedback while the mouse is tracked. See "Handling Mouse Events" in the Cookbook for further discussion of mouse trackers.
Call	Rarely

TrackMouse

```
FUNCTION TCommand.TrackMouse(aTrackPhase: TrackPhase; VAR anchorPoint, previousPoint,
                        nextPoint: VPoint; mouseDidMove: BOOLEAN): TCommand;
```

aTrackPhase	The current phase of the mouse-tracking process: trackPress when the mouse button first goes down, trackMove while the mouse moves, and trackRelease when the mouse button comes up
anchorPoint	The position of the mouse pointer, in view coordinates, when the mouse button went down. If you change this value, the new value is passed to you the next time this method is called.
previousPoint	The position of the mouse pointer the last time this method was called, in view coordinates
nextPoint	The current position of the mouse pointer, in view coordinates. Although you can change this value, it is better to use TrackConstrain to control mouse movement.
mouseDidMove	TRUE if the mouse moved since the last time TrackFeedback was called. (See "Track Feedback," below.)
The return value	The mouse tracker that will be used in succeeding calls. You generally return SELF, although applications may sometimes return a different mouse tracker object.

Purpose	To allow you to carry out any actions (other than feedback or mouse constraint) that depend on the movement of the mouse or on the track phase
Called by	TApplication.TrackMouse when the mouse button first goes down, as the mouse moves, and when the mouse button comes up
The default version	Returns SELF, in effect doing nothing
Override	Often. You override this method to take application-specific action. You should not assume that the mouse should be considered to have moved the first time this is called with an aTrackPhase of trackMove. The track phase is set to trackMove when the mouse moves more than the hysteresis value. SELF.TrackConstrain may set the mouse position back so that no movement should be considered to have occurred. The value of mouseDidMove should be tested to determine whether the mouse should be considered to have moved. See "Handling Mouse Events" in the Cookbook for further discussion of mouse trackers.
Call	Never

TList

Customize: rarely
Instantiate: often
Call methods: often

TList is defined in UList.

This object type is used in MacApp to store objects and is otherwise provided for your convenience. You do not have to use TList objects.

In general, you store objects of a single type in a TList object, and when you retrieve an object, you coerce the result into a variable of the type you need.

Ancestor: TObject

Fields

`fDeletions: INTEGER;`	The number of deleted elements in the list. These have the value kDeletedElement. The fSize field always reflects the number of real elements (that is, without counting these deleted elements). Other objects must not write directly to this field. (You can read its value, though.)
`fEachLevel: INTEGER;`	The number of Each calls in progress. Other objects must not write directly to this field.
`fFirstOffset: LONGINT;`	Contains the number of bytes of named fields before the first element. Equal to Sizeof(SELF). Other objects should neither read nor write to this field.
`fSize: INTEGER;`	Holds the number of elements in the list

At

```
FUNCTION TList.At(index: INTEGER): TObject;
```

Index	The index number of the element you want to retrieve (counting from one)

Purpose	To return a specific element from a list
The default version	Returns the requested element. Range checking is done only when the compile flag qRangeCheck is TRUE.
Override	Rarely
Call	Often

Delete

```
PROCEDURE TList.Delete(item: TObject);
```

item	A reference to an object
Purpose	To delete a specific element from the list
The default version	Searches the list for the first reference to the object referred to by item and deletes it. The item is not freed. If there are additional references to the same item in the list, they are not deleted. If the item is found, this method reduces fSize by one.
Override	Rarely. You might override this method to delete all references to the object referred to by item or to free the deleted object.
Call	Often

DeleteAll

```
PROCEDURE TList.DeleteAll;
```

Purpose	To delete all elements in a list
The default version	Sets fSize to 0. This deletes all elements from the list but does not free the objects.
Override	Rarely
Call	Often

Each

```
PROCEDURE TList.Each(PROCEDURE DoToItem(item: TObject));
```

DoToItem	A procedure (usually local) that is passed each element of the list in turn
Purpose	To apply the procedure DoToItem to every element in a list
The default version	Calls the procedure DoToItem repeatedly, passing each element of the list to that procedure in turn. The actual parameter is typically a procedure whose argument is a descendant of TObject. If DoToItem calls InsertLast, the newly added element will not be passed to DoToItem. If DoToItem calls InsertFirst or DeleteAll, the result is unpredictable.
Override	Rarely
Call	Often

First

```
FUNCTION TList.First: TObject;
```

The return value	The first element in the list
Purpose	To return the first element in a list
The default version	Returns the first element in the list, or NIL if there is no first element
Override	Rarely
Call	Often

FirstThat

```
FUNCTION TList.FirstThat(FUNCTION TestItem(item: TObject): BOOLEAN): TObject;
```

TestItem	A function, usually local to the caller, which returns TRUE when some condition is met
Purpose	To return the first element that fulfills some condition as determined by the function TestItem
The default version	Calls TestItem once for each element of the list, in order, until TestItem returns TRUE. It then completes and returns the element that satisfied the test. If none satisfied the test, the method returns NIL. The actual parameter is typically a function whose argument is a descendant of TObject. If TestItem calls InsertLast, the newly added element will not be enumerated. If TestItem calls InsertFirst, Delete, or DeleteAll, the results are unpredictable.
Override	Rarely
Call	Often

IList

```
PROCEDURE TList.IList;
```

Purpose	To initialize a new list
The default version	Initializes the list, setting fSize to 0
Override	Never. If you customize TList, you might supplement its action with an IYourList method.
Call	Usually. Sometimes, though, you call the global procedure NewList (documented with this object type, not in Chapter 9), which calls this method for you. You should never call this method twice for the same list.

InsertFirst

```
PROCEDURE TList.InsertFirst(item: TObject);
```

Item	An object reference
Purpose	To insert a new element as the first in a list
The default version	Inserts a reference to the item as the new first element of the list. The index of the new item is 1. All other elements are moved over one. (The old first element is not deleted; it is now the second element.) The value of fSize is increased by one. If the compile flag qDebug is TRUE and SetEltType was called, the item's type is checked to make sure it is of the list's defined element type. (That is only possible if the application and MacApp were compiled with debugging on. See Chapter 11 for more information.)
Override	Rarely
Call	Often

InsertLast

```
PROCEDURE TList.InsertLast(item: TObject);
```

Item	An object reference
Purpose	To insert a new element as the last element in a list
The default version	Inserts a reference to the item as the new last element of the list. The index of the new item is fSize. (The old last element is not deleted; it is now the next-to-last element.) The value of fSize is increased by one. If the compile flag qDebug is TRUE and SetEltType was called, the item's type is checked to make sure it is of the list's defined element type. (That is only possible if the application and MacApp were compiled with debugging on. See Chapter 11 for more information.)
Override	Rarely
Call	Often

NewList

```
FUNCTION NewList: TList;
```

Purpose	To create a linked list
The default version	Creates an object of type TList, calls IList to initialize it, and returns the object

* *Note*: This is a global procedure. It is documented here because it is important only for TList objects.

RemoveDeletions

```
PROCEDURE  TList.RemoveDeletions;
```

Purpose	To remove deleted items from a list. Items deleted by Delete while an Each operation is in progress are replaced by the value kDeletedElement. They cannot be accessed and are not counted in the value of fSize. This method actually removes those elements.
Override	Never
Call	Rarely